SEF - Week 5 Behaviour

Introduction

So far we have examined what users want the system to provide for them (use cases) and the data that will be used (classified into classes), and how they relate to each other.

We now turn back to describing more of the behaviour of the system - in this case how objects interact to achieve the use case goals (interaction diagrams) and internally (state diagrams - covered in a later lecture).

Interaction diagrams

UML has a number of interaction diagrams; the two we'll discuss are minor variations of each other.
All of the diagrams however attempt to show the interaction between objects - the messages they send to each other and the replies they give to achieve some goal. As such the diagrams represent the ***protocol*** that the objects follow.

Tip
We are all familiar with the protocol of talking on a phone. Typically the receiver identifies him/herself, the caller and callee then interact, until finally a signal is made that one person wants to end the phone call, and the call is terminated ("bye")).

A similar protocol exist for when an email client connects to a POP (Post Office Protocol) mail server to download or send email.

In both cases we can draw the participants and list the order and direction in which information (either a request or an answer) flows.
****

The diagrams are produced to give us a better understanding of how components in a system interact to accomplish a goal. Typically this directly represents how a Use Case goal occurs, but interaction diagrams can also be used to document how lower level components in a system interact.

Why do we want to document interactions?

This will help us understand how an existing system works, as well as showing the order in which communications can happen in a proposed system.

In both cases we get to observe the responsibilities of each object/entity/person in a system - we determine what role they are to play in meeting the use case goals.

An insurance company will, for example, have defined roles for their employees. An insurance assessor accepts insurance claims and decides whether to pay them. If the claim is for a house fire and it seems suspicious they will send a message to the fraud investigator who has the specialist skills required. The fraud investigator may send a message to the Fire Brigade who put out the fire for their assessment. The messages that flow back from each request inform each person and help determine what to do next.

We can identify which objects are the focus of many messages, and perhaps a suitable target for splitting into a number of roles/objects.

The diagrams can also be given to programmers who can use them when coding the system - they can follow the order of message sends and replicate it in the code they produce.

Communication and sequence diagrams.

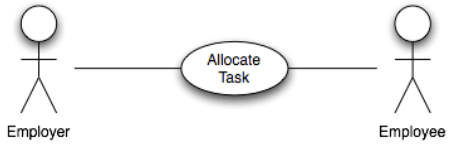
Communication and sequence diagrams show virtually the same information; the principle difference is in their visual layout.

Communication diagrams are essentially object diagrams which are annotated with messages flowing between the objects. The order of the messages is indicated numerically (1, 2, 2.1, 2.2, 3) etc. and is visually limiting when a large number of messages occur.

Sequence diagrams organise the objects across the page, with time running down the page.
This results in a diagram in which it is easier to see the order of the messages, but it has limitations when there are a large number of objects interacting.

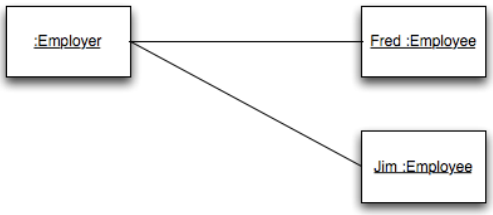
Communication diagrams

Consider the following use case "Allocate Task". In this diagram two actors are involved and we are going to examine how they interact (this is a very simple example, most diagrams involve multiple objects, and they are not necessarily all actors).

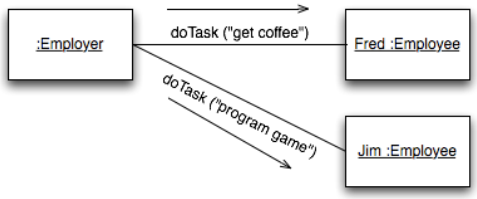


Actors are outside the scope of a system - they are exterior to it. Often however we want to internally model/mirror some information inside the system about the actors, and that's what we'll do here (a customer for example is external to a system, yet we may record lots of information about our customers).

Here we have both Employer and Employee as objects in an object diagram. The links between them could be "employs", or "gives instructions to".



Now we simply annotate the object diagram with message information, thus transforming it into a communication diagram!



We can see from this that one of the behaviours that an Employee has is "doTask". This is one of the services it provides to the system.
By examining all of the behaviours in the use cases - stepping through the various flow of events - you can identify the messages (requests for information, and answers) that each object is responsible for.

The reverse is also true - each service/responsibility we allocate to an object must ultimately be traceable back to a use case goal.

On what basis should services be allocated to objects?

We don't go to the local coffee shop to get advise on how to reindex an relational database, instead you would consult a specialist who (you hope!) has localised the knowledge pertinent to databases. Similarly we partition the behaviours and give them to objects that have the necessary attributes and associations to be able to act on those behaviours.

For example a Date object would be responsible for telling you what day of the week the date represents, or how many days separate it from another date. A database object would allow you to make queries and insert new data. A text editor object would provide a search feature.

Some objects have very little behaviour and may simply exist to store related attributes, although this tends to not be very common at all.

How can an object answer a request?

Either by examining its own attributes, or by asking other objects (that it knows about) on your behalf.
The example Date object described above can determine the day of the week it represents ("the 1st of October 2008 is a Wednesday") simply by examining its attributes of year, month and day.
It can determine how many days separate it from another date by examining its own y/m/d and asking the other date object about its attributes. In this case the second date is supplied as part of the request.

In other situations an object will consult other objects that it is associated with (via links, or associations in the class diagram). A request to a doctor object for the total value of the treatments/consultations provided would require it to request the value of the treatments and then to sum them and return the answer.

Scope of communication diagrams

Communication diagrams are about things communicating, but what are the things? The scope of the system will determine what those things are. For some systems the "things" are people - we can document the way people in an organisation communicate with each other to answer an incoming question.

The scope may also be a computer system - the "things" are simply objects/entities/variables/records/structs (whatever you want to call the data a program stores and manipulates) within the computer system. In this case what does communication mean? How would objects respond to a message?

A program is made up of lots of small code segments that do a single, relatively simple task called a method, or function. These perform some computation such as calculating what day of the week your birthday is on this year, or loading an image from a disk, or saving an appointment to a database. To do this they may interact with other objects to achieve the service.
To determine if there is an spare bed to admit a hospital patient you would need to examine each bed in the hospital to see if it is currently occupied. If you find that there is a spare bed, you would then send a message to bed object to mark itself as occupied. You can then send a message to the patient's record to note which bed they are currently allocated to.
This back and forth of communication to find answers, and set values is what the communication diagrams document.

Common mistakes in diagrams

A number of common mistakes made in diagrams are listed below.

Incorrect scope

The purpose of an interaction diagram is to show how the objects in the system you are modelling will interact. There is no point showing the interactions between objects that are outside the scope of the system.

If you are modelling a computer system (and not the larger organisation that will use the computer system) your objects need to be thought of solely as computer objects, not as people who will use the system.

A common mistake is to confuse an Actor (such as Customer) with the Customer object that records information about them. A customer actor is an

independent entity that can initiate all sorts of behaviours. A customer object is likely to be a more passive object that only responds to requests without initiating any of its own.

Requesting information from an object that can't supply it

Behaviours can be allocated to objects, but there's not much point in doing so if they don't have the attributes or associations to find out.
As discussed above the local coffee shop knowledge is constrained to making coffees, nor would they be able to refer you on to anyone who would be able to answer your database queries.

Requesting services from objects that are not reachable

The associations that are listed in a class diagram are a little like phone numbers - it says that the two objects remember each other when the system is operational. This allows them to communicate. If there were no association, it wouldn't be possible for one object to "phone" another object when it needs a question answered - it simply wouldn't know who to call.

Very often the behaviours allocated to objects are "lookup" services - they will provide the phone number of another object so you can have a conversation with them.

For example a message may need to a plane that is closest to an airport. The system could request a radar system to search through the list of planes that it is tracking and return the "phone number" of the closest aircraft, allowing the object originating the request to then converse directly with the aircraft.